【物理模拟】PBD算法详解

blog.csdn.net 2022-07-30 22:58

参考： Matthia Muller的十分钟物理（他就是PBD算法的发明者） https://matthias-research.github.io/pages/tenMinutePhysics/

PBD的算法主体分为三步：

由于第3步是先求出粒子位置，再反求速度的。因此它是基于位置的方法，故被称为position based dynamics。

PBD是纯粒子法。在PBD的世界中，只有粒子。其他的一切（三角面、四面体等）都是辅助性的。

while simulating
  for all particles i
    $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\ \mathbf{g}$
    $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t\ \mathbf{v}_i$

  for all constraints C
    solve$(C, \Delta t)$

  for all particles i
    $\mathbf{v}_i \leftarrow (\mathbf{x}_i - \mathbf{p}_i)/\Delta t$

solve$(C, \Delta t)$:

for all particles i of C
    compute $\Delta\mathbf{x}_i$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta\mathbf{x}_i$

算法主体

如图，先根据外力更新速度（这里只考虑了重力）

然后把旧的位置存到p中

最后利用速度更新位置

（如果有碰撞，也在这里处理）

整个过程粒子无需考虑相互关系。

for all particles i
  $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\ \mathbf{g}$
  $\mathbf{p}_i \leftarrow \mathbf{x}_i$

图片加载中...

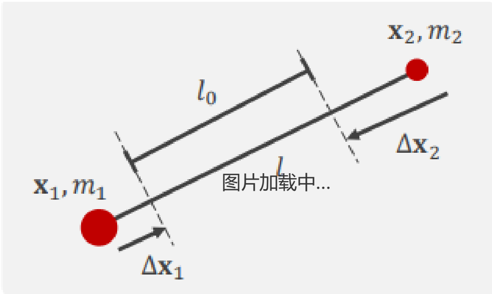
$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t\ \mathbf{v}_i$$

在这里插入图片描述

我们分两种情况：一种是最简单的二维情况，一种是三维情况。

我们先从简单的二维情况来：

我们只考虑一个弹簧的约束



在这里插入图片描述

这个弹簧只有两个质点和中间的一个边。质点具有位置和质量，边具有原长度和现长度。

随意移动两个粒子，弹簧目前不处于原长。

因此，弹簧要回到原长。

弹簧回到原长，这就是这个系统的约束。

C (x 1 , x 2) = ∣ x 1 − x 2 ∣ − L 0 C(x1, x2) = |x1 - x2| - L0 C(x1,x2)=∣x1−x2∣−L0 其中x1, x2分别是粒子1和粒子2的位置。L0是原长。

如何让系统满足约束呢?

所谓的满足约束，就是让约束误差等于0。

上面这个式子中的C，就是约束的误差（有时候约束和约束误差这两个词混用）。

通过迭代，让误差趋于0，那就是求解过程。

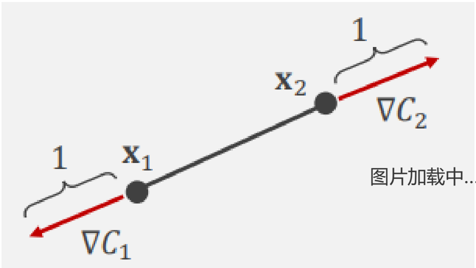
如何让其趋于0?

那就是求梯度，然后向着梯度减小的方向移动（即所谓的梯度下降思想）。

实际上，从另一个角度来看，梯度就是一维的导数在高维的推广。让导数等于0的位置（即驻点），就是原函数最小化的位置。

于是我们就找C的梯度。

我们无需那样严谨地去推导数学公式，然后给出C梯度的表达式。我们直接从物理意义上来理解C梯度。那就是让C函数上涨最快的方向。在这个例子中，也就是x1与x2相对的方向。因此我们给出上面的公式。而且我们这里是归一化了的，只求出一个方向。



$$\nabla C_1 = \frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$

$$\nabla C_2 = \frac{\mathbf{x}_2 - \mathbf{x}_1}{|\mathbf{x}_2 - \mathbf{x}_1|}$$

图片加载中...

在这里插入图片描述

那么大小是多少呢？我们给出如下公式

$$\lambda = \frac{-C}{w_1|\nabla C_1|^2 + w_2|\nabla C_2|^2 + \cdots + w_n|\nabla C_n|^2} = \frac{-(l - l_0)}{w_1 \cdot 1 + w_2 \cdot 1}$$

$$\Delta \mathbf{x}_1 = \lambda\, w_1\, \nabla C_1 = -\frac{w_1}{w_1 + w_2}(l - l_0)\frac{\mathbf{x}_2 - \mathbf{x}_1}{|\mathbf{x}_2 - \mathbf{x}_1|}$$

图片加载中...

在这里插入图片描述

大小是由系数lambda和质量倒数w决定的。其中lambda的正式名称叫做拉格朗日乘数。

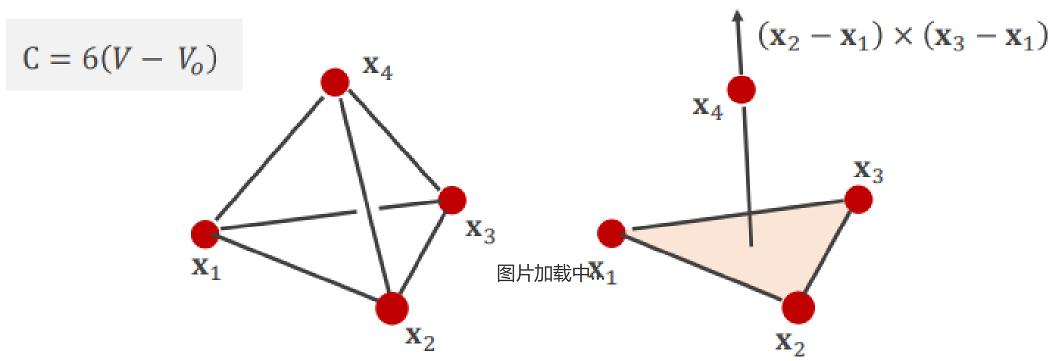
那么，我们求解出来了gradC，因此就求解出来了粒子间的相互关系。因此，也就知道粒子为了满足相互关系，该向哪里移动。因此给出了dx（如上图）。这个移动的方向，就是最小化约束误差的方向，就是负梯度的方向（因此lambda分母有个负号）。

对于三维，我们期望的约束是四面体的体积保持原体积。

其约束误差就是 C = (V − V 0) C = (V - V0) C=(V−V0)

而四面体的体积公式是 1 6 [(x 2 − x 1) × (x 3 − x 1)] ⋅ (x 4 − x 1) \frac{1}{6} [(x2 - x1)\times (x3 - x1)] \cdot (x4 - x1) 61[(x2−x1)×(x3−x1)]⋅(x4

−x1) 先叉乘求出底面积（叉乘得到的是菱形面积，还要除以2），然后再乘以高度（点乘会消除非垂直部分），再乘以1/3(四面体公式中本来的1/3)



$$C = 6(V - V_o) = [(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)] \cdot (\mathbf{x}_4 - \mathbf{x}_1) \; - 6V_o$$

$$\nabla_4 C = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)$$

在这里插入图片描述

而约束误差的梯度是什么呢？我们仍然跳过数学推导，从物理意义上解释。

梯度即函数增长最快的方向，也就是体积增长最快的方向。对于某个粒子来说，哪个方向让体积增长最快呢？

那就是垂直于底面的方向。

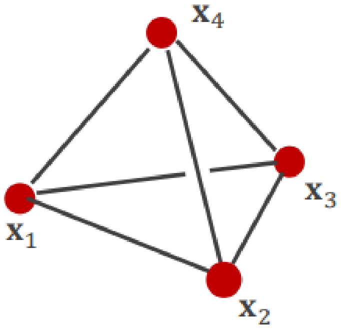
而垂直与底面的方向，就是底面的三角形其中两个边叉乘的方向（叉乘满足右手定则）。

因此 g r a d C 4 = (x 2 − x 1) × (x 3 − x 1) gradC4 = (x2 - x1) \times (x3 - x1) gradC4=(x2−x1)×(x3−x1) 这就是粒子4所对应约束的梯度。

那么大小如何确定呢？

仍然利用拉格朗日乘数lambda。

Solve



Right hand rule

$$\nabla_1 C = (\mathbf{x}_4 - \mathbf{x}_2) \times (\mathbf{x}_3 - \mathbf{x}_2)$$
$$\nabla_2 C = (\mathbf{x}_3 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_1)$$
$$\nabla_3 C = (\mathbf{x}_4 - \mathbf{x}_1) \times (\mathbf{x}_2 - \mathbf{x}_1)$$

图片加载中...

$$\nabla_4 C = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)$$

$$\lambda = \frac{-6(V - V_o)}{w_1|\nabla C_1|^2 + w_2|\nabla C_2|^2 + w_3|\nabla C_3|^2 + w_4|\nabla C_4|^2}$$

$$\Delta \mathbf{x}_i = \lambda\, w_i\, \nabla C_i$$

在这里插入图片描述

值得注意的是： 求解一个三维弹性物体，我们必须同时满足弹簧约束和体积约束。体积约束保证弹性体不发生体积的膨胀和缩小，而弹簧约束保证物体上的每个质点回到原位（也就是最终弹性体恢复原状）。

我们参考的是 https://matthias-research.github.io/pages/tenMinutePhysics/ 中第10讲的代码

数据结构： 存储在class SoftBody内。SoftBody可多次实例化以添加多个物体。

simulate函数为算法的主体，它分为三步：

又被分为两个部分：

也就对应上面说的弹簧约束和体积约束。

拷贝自tenMinutePhysics

```
preSolve(dt, gravity)
{
    for (var i = 0; i < this.numParticles; i++) {
        if (this.invMass[i] == 0.0)
            continue;
        vecAdd(this.vel,i, gravity,0, dt);
        vecCopy(this.prevPos,i, this.pos,i);
        vecAdd(this.pos,i, this.vel,i, dt);
        var y = this.pos[3 * i + 1];
        if (y < 0.0) {
            vecCopy(this.pos,i, this.prevPos,i);
            this.pos[3 * i + 1] = 0.0;
        }
    }
}
```

在这里插入图片描述

```
solveEdges(compliance, dt) {
    var alpha = compliance / dt /dt;

    for (var i = 0; i < this.edgeLengths.length; i++) {
        var id0 = this.edgeIds[2 * i];
        var id1 = this.edgeIds[2 * i + 1];
        var w0 = this.invMass[id0];
        var w1 = this.invMass[id1];
        var w = w0 + w1;
        if (w == 0.0)
            continue;

        vecSetDiff(this.grads,0, this.pos,id0, this.pos,id1);
        var len = Math.sqrt(vecLengthSquared(this.grads,0));
        if (len == 0.0)
            continue;
        vecScale(this.grads,0, 1.0 / len);
        var restLen = this.edgeLengths[i];
        var C = len - restLen;
        var s = -C / (w + alpha);
        vecAdd(this.pos,id0, this.grads,0, s * w0);
        vecAdd(this.pos,id1, this.grads,0, -s * w1);
    }
}
```

在这里插入图片描述

```
solveVolumes(compliance, dt) {
    var alpha = compliance / dt /dt;

    for (var i = 0; i < this.numTets; i++) {
        var w = 0.0;

        for (var j = 0; j < 4; j++) {
            var id0 = this.tetIds[4 * i + this.volIdOrder[j][0]];
            var id1 = this.tetIds[4 * i + this.volIdOrder[j][1]];
            var id2 = this.tetIds[4 * i + this.volIdOrder[j][2]];

            vecSetDiff(this.temp,0, this.pos,id1, this.pos,id0);
            vecSetDiff(this.temp,1, this.pos,id2, this.pos,id0);
            vecSetCross(this.grads,j, this.temp,0, this.temp,1);
            vecScale(this.grads,j, 1.0/6.0);

            w += this.invMass[this.tetIds[4 * i + j]] * vecLengthSquared(this.grads,j);
        }
        if (w == 0.0)
            continue;

        var vol = this.getTetVolume(i);
        var restVol = this.restVol[i];
        var C = vol - restVol;
        var s = -C / (w + alpha);

        for (var j = 0; j < 4; j++) {
            var id = this.tetIds[4 * i + j];
            vecAdd(this.pos,id, this.grads,j, s * this.invMass[id])
        }
    }
}
```

在这里插入图片描述

```
postSolve(dt)
{
    for (var i = 0; i < this.numParticles; i++) {
        if (this.invMass[i] == 0.0)
            continue;
        vecSetDiff(this.vel,i, this.pos,i, this.prevPos,i, 1.0 / dt);
    }
    this.updateMeshes();
}
```

在这里插入图片描述